CerFix: A System for Cleaning Data with Certain Fixes

Wenfei Fan [1,2] Jianzhong Li[2] Shuai Ma[3] Nan Tang[1] Wenyuan Yu[1]
[1]University of Edinburgh [2]Harbin Institute of Technology [3]Beihang University
{wenfei@inf., ntang@inf., wenyuan.yu@}ed.ac.uk lijzh@hit.edu.cn mashuai@act.buaa.edu.cn

Abstract

We present CerFix, a data cleaning system that finds *certain fixes* for tuples at the point of data entry, *i.e.,* fixes that are guaranteed correct. It is based on master data, editing rules and certain regions. Given some attributes of an input tuple that are validated (assured correct), editing rules tell us what other attributes to fix and how to correct them with master data. A certain region is a set of attributes that, if validated, warrant a certain fix for the entire tuple. We demonstrate the following facilities provided by CerFix: (1) a region finder to identify certain regions; (2) a data monitor to find certain fixes for input tuples, by guiding users to validate a minimal number of attributes; and (3) an auditing module to show what attributes are fixed and where the correct values come from.

1. Introduction

It has long been recognized that real-life data is often dirty [10]. Dirty data is costly: it costs US businesses 600 billion dollars each year [5]. This highlights the need for data cleaning tools, to help users detect errors in the data and moreover, repair the data, *i.e.,* correct the errors.

Most data cleaning systems on the market are ETL tools (extraction, transformation, loading; see [8] for a survey). To detect semantic errors in the data, there have also been a host of approaches on data repairing based on integrity constraints [1, 2, 4, 11]. A variety of constraints have been studied for data repairing, such as traditional functional, inclusion and full dependencies [1, 2], as well as their extensions (*e.g.,* conditional functional dependencies [4, 11]).

Integrity constraints are capable of detecting the presence of errors in the data, *i.e.,* determining whether the data is dirty or not. However, they do not tell us which attributes of a tuple have errors and how we could correct the errors.

Example 1: Consider an input tuple:

t: (FN = Bob, LN =Brady, AC = 020, phn =079172485, type =2, str = 501 Elm St, city = Edi, zip = EH8 4AH, item = CD)

It specifies a UK customer: name (FN, LN), phone number (area code AC and phone phn), address (street str, city, zip code) and item purchased. Here phn is either home phone or mobile phone, indicated by type (1 or 2, respectively).

Proceedings of the VLDB Endowment, Vol. 4, No. 12

To specify the semantics of the input data, one may define the following conditional functional dependencies (CFDs):

ψ_1: AC = 020 → city = Ldn, ψ_2: AC = 131 → city = Edi.

These CFDs state that if AC is 020, then city should be Ldn (ψ_1), and when AC is 131, then city must be Edi (ψ_2). They detect the presence of errors in tuple t: t[AC] = 020 but t[city] = Edi. However, they do not tell us which of t[AC] and t[city] is wrong, and to what value it should be changed.

In light of this, previous constraint-based methods use *heuristics*: they do not guarantee correct fixes in data repairing. Worse still, they may introduce new errors when trying to repair the data. Indeed, the correct values of t[AC, city] are (131, Edi). Indeed, all these previous methods may opt to change t[city] to Ldn; this does not fix the erroneous t[AC] and worse, messes up the correct attribute t[city]. □

In practice, one often wants to find *certain fixes* for the data, *i.e.,* fixes that are guaranteed to be correct. In other words, if a data repairing process changes the value of an attribute $t[A]$ to v, then v should be the true value of $t[A]$. In addition, it should ensure that no new errors are introduced in the process. The need for certain fixes is particularly evident when repairing critical data, *e.g.,* medical data, in which a seemingly minor error may mean life or death.

An approach to finding certain fixes was proposed in [7], based on master data, editing rules and certain regions. *Master data* (*a.k.a. reference data*) is a single repository of high-quality data that provides various applications in an enterprise with a synchronized, consistent view of its core business entities [9]. It is increasingly common for enterprises to maintain master data nowadays, and master data management (MDM) is being developed by IBM, SAP, Microsoft and Oracle. In contrast to integrity constraints, *editing rules* aim to tell us which attributes of a tuple are wrong and what values from master data they should take, provided that master data is available and that some other attributes are validated (assured correct). With respect to a set of editing rules and a master relation, a *certain region* is a set $\mathcal{A}$ of attributes and a pattern tableau, such that for any input tuple t, if $t[\mathcal{A}]$ is validated and $t[\mathcal{A}]$ bears a pattern in the tableau, then the editing rules guarantee to find a (unique) certain fix for all the attributes of t.

Example 2: Suppose that master data contains a tuple:

s: (FN = Robert,¡ LN =Brady,¡ AC = 131, Hphn = 6884563, Mphn =079172485, str = 501 Elm St, city = Edi, zip = EH8 4AH, DoB = 11/11/55, gender = M)

An editing rule is given as follows:

φ_1: ((zip, zip) → (AC, AC), t_{p_1} = ())

It states that for an input tuple t and a master tuple s, if they have identical zip code (*i.e.,* t[zip] = s[zip]), then t could be updated by t[AC] := s[AC], provided that t[zip] is

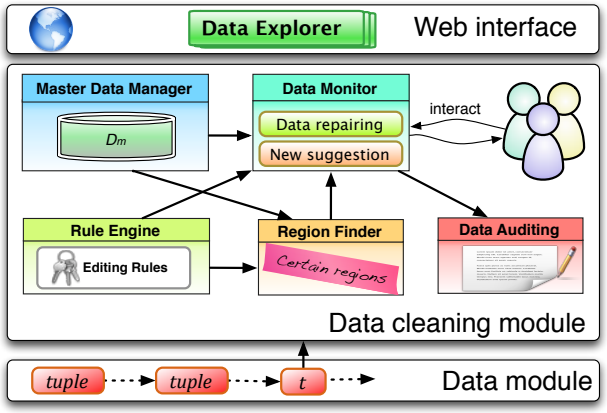


Figure 1: The CerFix Architecture.

correct (either ensured by the users or verified automatically). When the master data is accurate, the editing rule is assured valid, and the region $t[\mathsf{zip}]$ is correct, the fix to $t[\mathsf{AC}]$ with the master data value is certainly correct. □

Following [7], we develop CerFix, a data cleaning system that finds certain fixes for input tuples at the point of data entry. It differs from other systems that also ask for user feedback (*e.g.,* [3]) in what feedback is requested and how the feedback is used. Below we first present CerFix, and then outline what functionalities we shall demonstrate.

2. The CerFix System

The architecture of CerFix is depicted in Fig. 1. CerFix maintains a collection of master data (master data manager) and a set of editing rules (rule engine). With respect to the master data and editing rules, it computes a set of certain regions (region finder). It inspects and repairs input tuples via a data monitor, which interacts with the users to find certain fixes for input tuples. It also provides a Web interface (data explorer) and a data auditing module for users to manage editing rules and trace changes to data, respectively.

Below we briefly present the key components of CerFix. We refer the interested readers to [7] for details about editing rules, certain regions and their computation.

Rule engine. It maintains a set of editing rules (eRs) that specify (a) whether an input tuple t can match a master tuple s via a pattern tuple (*e.g.,* $(\mathsf{zip}, \mathsf{zip})$ together with t_{p_1} in rule φ_1 of Example 2), and (b) which attribute values of t can be changed and what correct values they should take from the master data. The engine also implements static analysis techniques developed in [7]. In particular, it supports the following. (1) It checks the consistency of editing rules, *i.e.,* whether the given rules are dirty themselves. Moreover, (2) provided that some attributes of a tuple are correct, it automatically derives what other attributes can be validated (assured correct) by using editing rules and master data.

Editing rules can be either explicitly specified by the users, or derived from integrity constraints, *e.g.,* CFDs and matching dependencies [6] for which discovery algorithms are already in place. CerFix currently only supports manual specification of editing rules via the Web interface.

Master data manager. It maintains master data, which is assumed consistent and accurate [9].

Region finder. A region is a pair (Z, T_c), where Z is a list of attributes of an input tuple and T_c is a *pattern tableau* consisting of a set of pattern tuples with attributes in Z.

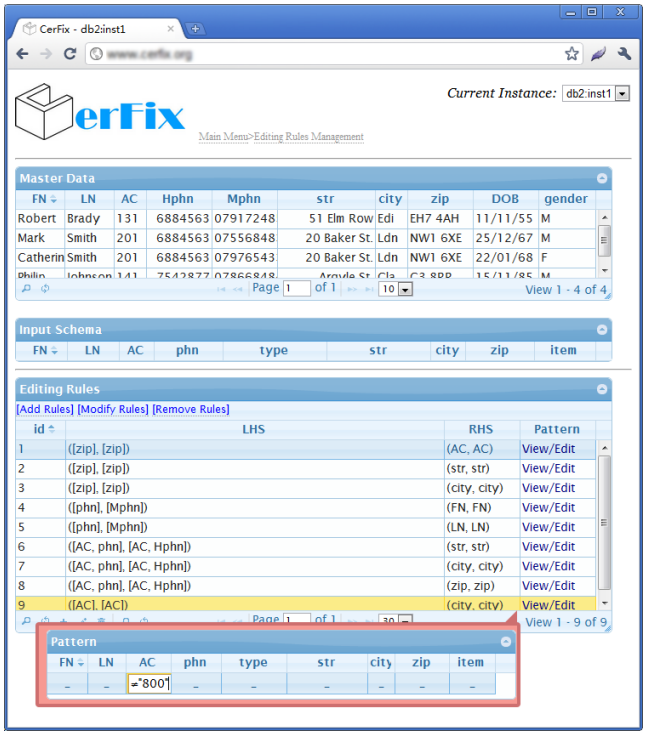


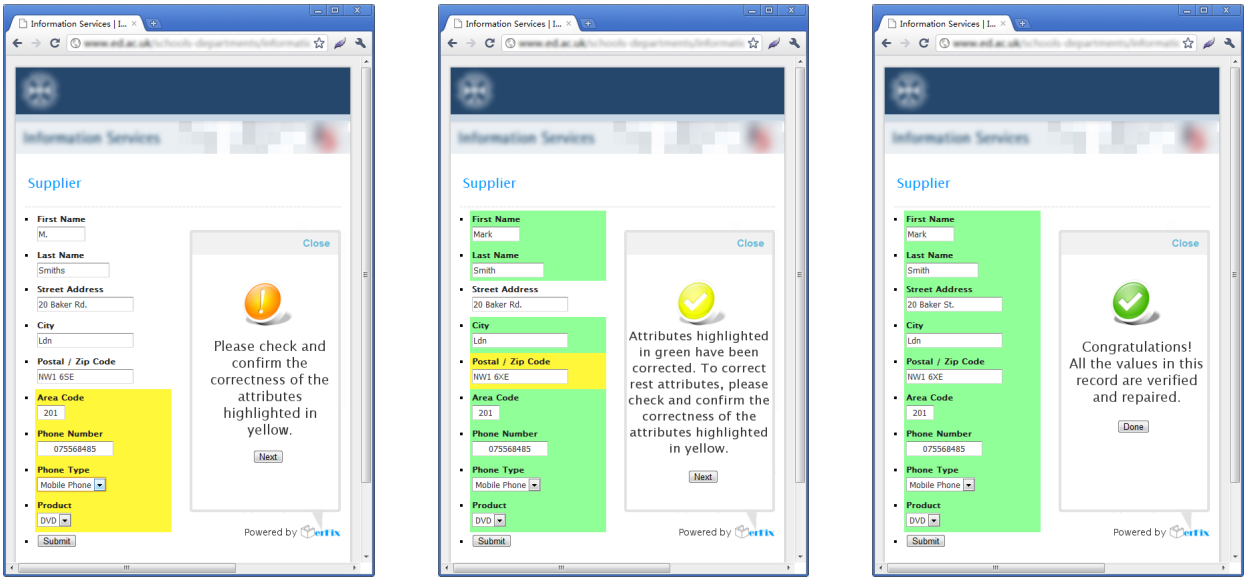
Figure 2: Management of editing rules.

A region (Z, T_c) is a *certain region w.r.t.* a set of editing rules and master data if for any input tuple t, as long as $t[Z]$ is correct and $t[Z]$ matches a pattern in T_c, the editing rules warrant to find a certain fix for t. Based on the algorithms in [7], top-k certain regions are pre-computed that are ranked ascendingly by the number of attributes, and are recommended to users as (initial) suggestions.

Data monitor. This is the most important module of CerFix. It interacts with the users and finds certain fixes for input tuples, as follows. (1) *Initial suggestions.* It recommends the set of certain regions computed by region finder to the users as suggestions. For each input tuple t, if the users ensure that $t[Z]$ is correct and matches a pattern in T_c for any region (Z, T_c) in the set, then a certain repair for t is warranted. (2) *Data repairing.* For an input tuple t, the users may respond with a set $t[S]$ of attributes that is correct, where S may *not* be any of the certain regions. Data monitor iteratively employs editing rules and master data to fix as many attributes in t as possible, and expands the correct attribute set S by including those attributes that are validated via the inference system of the rule engine. (3) *New suggestion.* If not all attributes of t have been validated, data monitor computes a new suggestion, *i.e.,* a *minimal* number of attributes, which are recommended to the users. If the users ensure the correctness of these attributes in t, data monitor will find a certain fix for t. The process of steps (2) and (3) repeats until a certain fix of t is reached.

CerFix ensures that each fix is correct with editing rules and master data. It also minimizes users' effort by identifying a minimal number of attributes for users to validate.

Data auditing. This module keeps track of changes to each tuple, incurred either by the users or automatically by data monitor with editing rules and master data. Statistics about the changes can be retrieved upon users' requests.



(a) Providing suggestions (b) Fixing data (c) Achieving certain fixes

Figure 3: Data monitor.

3. Demonstration Overview

We next describe various aspects of CerFix in more detail, and explain the aims of our demonstration. More specifically, we show the following: (1) how users manage editing rules with the aid of the Web interface (data explorer in Fig. 1); (2) how CerFix interacts with the users for data monitoring, to detect and fix errors in input tuples at the point of their entry; and (3) how data auditing works, to keep track of which attributes are fixed and where the correct values come from, and provide statistics about the percentage of data that is fixed by user efforts or by CerFix.

Initialization. The users are required to configure an instance, which consists of two parts: (a) a data connection with JDBC url, username, password and the corresponding JDBC driver provided by users; and (b) specifying the schema of input (dirty) tuples and that of the master data.

We illustrate these with the master data and the input data shown in Fig. 2. Note that they have different schemas.

Master data. Each tuple in the master data specifies a person in the UK in terms of the name (FN, LN), area code (AC), home phone (Hphn), mobile phone (Mphn), address (street str, city and zip code), date of birth (DOB) and gender.

Input tuples. Each tuple specifies a UK customer, as illustrated in Example 1.

Editing rule management. Figure 2 displays the Web interface for managing eRs. In the demonstration we shall show how the users can manage (view/modify/add/delete) eRs using the data explorer. The system currently only supports to import eRs manually via the rule manager, where the eRs may either be designed by experts or be discovered from CFDs or MDs. For instance, Figure 2 shows nine editing rules φ_1–φ_9, for the id's 1–9, respectively.

- φ_1 (resp. φ_2 and φ_3) states that if an input tuple t and a master tuple s have the same zip code and if t[zip] is already validated, then t can be updated by t[zip] := s[zip] (resp. str and city).
- φ_4 (resp. φ_5) states that if the phn of a tuple t matches the Mphn of a master tuple s, and if t[phn] is validated, then t[FN] := s[FN] (resp. LN). These eRs pose a constraint (a pattern tuple) t[type] = 2, requiring that phn is mobile phone. This rule can be viewed or edited by clicking the view/edit frame for the pattern.
- φ_6 (resp. φ_7 and φ_8) tells us that if the (AC, phn) attributes of an input tuple t match the (AC, Hphn) values of a master tuple s, and if t[AC, phn] are validated, then t[str] := s[str] (resp. city and zip). These eRs have a pattern t[type] = 1, *i.e.,* phn is home phone.
- φ_9 states that when the AC value of an input tuple t is not 0800 (toll free, non-geographic), if it agrees with a master tuple s on its AC attribute, and moreover, if t[AC] has been validated, then t should be updated with t[city] := s[city]. As shown in Fig. 2, the pattern "$\neq$ 0800" can be edited via a pop-up frame.

CerFix automatically tests whether the specified eRs make sense *w.r.t.* master data, *i.e.,* the rules do not contradict each other and will lead to a *unique* fix for any input tuple. Furthermore, given certain attributes that are validated, it automatically derives what other attributes can be validated by eR and master data, via an inference system.

Data monitor. We shall demonstrate how CerFix interacts with users to find a certain fix for each input tuple.

1. CerFix suggests a set of attributes for the users to validate. The users may either validate these attributes, or opt to assure that some other attributes are correct. The initial suggestions are computed by region finder.
   - As shown in Fig. 3(a), the attributes suggested by CerFix are highlighted in yellow, *i.e.,* area code AC, phone number phn, phone type, and product item. The values of the attributes assigned by the users are 201, 075568485, Mobile phone (type 2), and DVD, respectively.
2. If the users opt to validate these attributes, CerFix iteratively applies editing rules and master data to the data, and expands the set of attributes validated.
   - As shown in Fig. 3(b), all attributes that have been validated are now highlighted in green. These include attributes first name FN, last name

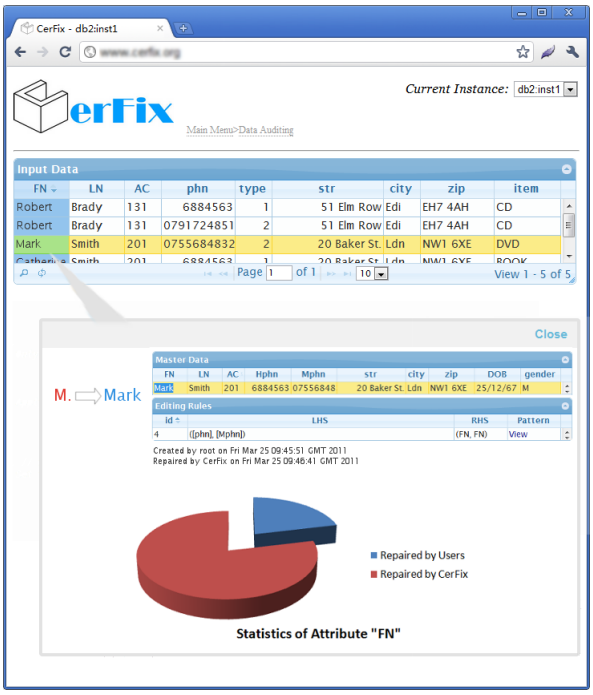


Figure 4: Data auditing.

LN, and city, for which the correctness is validated by CerFix. For instance, the value of FN is normalized from 'M.' to 'Mark' by eR φ_4 with the FN value of the second master tuple in Fig. 2.

The users may decide to validate attributes other than those suggested. CerFix reacts by fixing data with editing rules and master data in the same way, based on the attributes selected and validated by the users.

3. If some attributes of the input tuple are still not validated, CerFix computes a new suggestion and goes back to step 1, to interact with the users by providing the new suggestion. In each interaction, both the users and CerFix expand the attributes that are validated.
   - As shown in Fig. 3(b), CerFix suggests the users to validate zip code. After two rounds of interactions, all the attributes are validated. This is shown in Fig. 3(c) with all attributes in green.

When fixing the data, the most time-consuming procedure is to compute suggestions. To reduce the cost, CerFix precomputes a set of certain regions with *region finder* (see Fig. 1), which are provided to the users as initial suggestions, and are referenced when computing new suggestions.

We remark that data monitor of CerFix is quite generic, *i.e.,* it does not depend on any particular system. Indeed, it supports several interfaces to access data, which could be readily integrated with other database applications.

Data auditing. CerFix provides a data auditing facility such that after a stream of input tuples is fixed, the users may inspect the changes made to those tuples.

The users may inspect attributes of an individual tuple. For instance, as shown in Fig. 4, when the users select the FN attribute of a tuple (highlighted in yellow), CerFix shows that it has been fixed by normalizing the first name 'M.' to 'Mark'. It further presents what master tuples and editing rules have been employed to make the change.

The users may also want to inspect each attribute (column) of the input tuples. As shown in Fig. 4, when FN is selected, CerFix presents the statistics about the attribute FN, namely, the percentage of FN values that were validated by the users and the percentage of values that were automatically fixed by CerFix. Our experimental study indicates that in average, 20% of values are validated by users while CerFix automatically fixes 80% of the data.

Summary. The demonstration aims to exhibit the strength of editing rules and important functionalities of CerFix. (1) Editing rules. As opposed to integrity constraints that only detect the presence of errors in the data, editing rules identify what attributes are erroneous and tell us how to correct the errors with master data. (2) Region finder. It tells us to validate an input tuple, what minimal sets of attributes have to be assured correct. (3) Data monitor. It interacts with the users to find certain fixes, while minimizing human efforts by suggesting a minimal number of attributes for the users to validate. (4) Data auditing. It helps the users understand better the quality of input data sets.

Acknowledgments. Fan is supported in part by the RSE-NSFC Joint Project Scheme and an IBM scalable data analytics for a smarter planet innovation award. Li is supported in part by NGFR 973 grant 2006CB303000 and NSFC grant 60533110. Shuai is supported in part by NGFR 973 grant 2011CB302602 and NSFC grants 90818028 and 60903149.

4. References

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[2] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.

[3] K. Chen, H. Chen, N. Conway, J. M. Hellerstein, and T. S. Parikh. Usher: Improving data quality with dynamic forms. In *ICDE*, pages 321–332, 2010.

[4] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326, 2007.

[5] W. W. Eckerson. Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data. In *The Data Warehousing Institute*, 2002.

[6] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, pages 407–418, 2009.

[7] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, pages 173–184, 2010.

[8] T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data Quality and Record Linkage Techniques*. Springer, 2009.

[9] D. Loshin. *Master Data Management*. Knowledge Integrity, Inc., 2009.

[10] T. Redman. The impact of poor data quality on the typical enterprise. *Commun. ACM*, 2:79–82, 1998.

[11] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, pages 279–289, 2011.